Flow-Service-Quality (FSQ) Engineering: Foundations for Network System Analysis and Development

Richard C. Linger
Mark G. Pleszkoch
Gwendolyn Walton
Alan R. Hevner

June 2002

TECHNICAL NOTE
CMU/SEI-2002-TN-019

Carnegie Mellon
Software Engineering Institute

Pittsburgh, PA 15213-3890

Flow-Service-Quality (FSQ) Engineering: Foundations for Network System Analysis and Development

CMU/SEI-2002-TN-019

Richard C. Linger
Mark G. Pleszkoch
Gwendolyn Walton
Alan R. Hevner

June 2002

Networked Systems Survivability Program

The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

NO WARRANTY

THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

Use of any trademarks in this report is not intended in any way to infringe on the rights of the trademark holder.

This work was created in the performance of Federal Government Contract Number F19628-00-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 252.227-7013.

For information about purchasing paper copies of SEI reports, please visit the publications portion of our Web site (http://www.sei.cmu.edu/publications/pubweb.html).

Table of Contents

Abstract..v

1 Network System Realities...1

2 Flow-Service-Quality Engineering...5

3 Flow Structure Semantics...11

3.1 The Semantic Model 11

3.2 FSQ Theorems..15

4 Flow Structure Engineering Operations 19

4.1 Flow Engineering for Uncertainty Factors................................19

4.2 Flow Abstraction and Refinement..20

4.3 Flow Verification 22

4.4 Flow Transitivity Analysis..23

4.5 FlowSets in Large-Scale Systems 24

4.6 Flow Security and Survivability Analysis 25

5 Computational Quality Attributes..27

5.1 The CQA approach...27

5.2 CQA Definition..29

5.3 Flow Request Analysis 29

5.4 A CQA Example 31

5.5 Considerations for CQA Analysis...32

5.6 Dynamic Updates of Computational Quality Attributes.............33

6 Flow Management Architectures..35

7 Conclusion...37

8 References...39

List of Figures

Figure 1: Elements of the Network-Centric Future Combat System........................1

Figure 2: System-of-Systems Traversals in a Gasoline Purchase Transaction........2

Figure 3: Refinement into User Task Flows into System Service Uses....................6

Figure 4: FSQ Engineering Operations for New and Existing Systems 8

Figure 5: Superimposing a Deterministic Flow on an Asynchronous Network 8

Figure 6: Elements of Flow-Service Semantics 12

Figure 7: Typical FSL Control Structures...14

Figure 8: Post-Fix Service Response Evaluation for Survivability Analysis and Risk Management..20

Figure 9: Algebraic Operations in Flow Structure Analysis and Design..................21

Figure 10: Flow Abstraction and Refinement in a Transaction-Based System.........21

Figure 11: The Correctness Evaluation Process Based on the Flow Verification Theorem 22

Figure 12: Transitivity Analysis of Flow Dependencies...24

Figure 13: Flow-Sets for the Future Combat System..25

Figure 14: Flow Security and Survivability Domain Traversals 26

Figure 15: The Computational Quality Attribute Approach.......................................28

Figure 16: Simple Constrained Flow Request 31

Figure 17: State Diagram for Analysis of q_1 32

Abstract

Modern society could hardly function without the large-scale, network-centric information systems that pervade government, defense, and industry. As a result, serious failures or compromises carry far-reaching consequences. These systems are characterized by changing and often unknown boundaries and components, constantly varying function and usage, and complexities of pervasive asynchronous operations. Their complexity challenges human intellectual control, and their survivability has become an urgent priority. Engineering methods based on solid foundations and the realities of network systems are required to manage complexity and ensure survivability. Flow-Service-Quality (FSQ) engineering is an emerging technology for management, acquisition, analysis, development, evolution, and operation of large-scale, network-centric systems. FSQ engineering is based on Flow Structures, Computational Quality Attributes, and Flow Management Architectures. These technologies can help provide stable engineering foundations for the dynamic and often unpredictable world of large-scale, network-centric systems. Flow Structures define enterprise mission task flows and their refinements into uses of system services in network traversals. Flows are deterministic for human understanding, despite the underlying asynchronism of network operations. They can be refined, abstracted, and verified with precision, and deal explicitly with Uncertainty Factors, including uncertain commercial off-the-shelf functionality and system failures and compromises. Computational Quality Attributes go beyond static, a priori estimates to treat quality attributes such as reliability and survivability as dynamic functions to be computed in system operation. Computational Quality Attribute requirements are associated with flows and can be dynamically reconciled with network service attributes in execution. Flow Management Architectures include design and implementation frameworks for dynamically managing flows and attribute requirements, as well as processes for their development. FSQ foundations are defined by theorems that illuminate engineering practices and automation opportunities.

1 Network System Realities

Modern enterprises are irreversibly dependent on large-scale network systems whose complexity frequently exceeds current engineering capabilities for intellectual control. The result has been persistent difficulties in development, management, and evolution, and failures, intrusions, and compromises in operation [Schneider 1999]. These systems are characterized by very-large-scale heterogeneous networks with often-unknown boundaries and components. Dynamic interconnectivity of systems-of-systems can limit visibility and control of security and survivability. User task flows can traverse systems and boundaries with varying security and survivability characteristics. In addition, these systems must deal with uncertain commercial off-the-shelf (COTS) function and quality, unforeseen behaviors and vulnerabilities, and unanticipated inter-system cascade failures. Complexity is compounded by the extensive asynchronous behavior of the virtually unknowable interleaving of communications among system components. Figure 1 depicts components of such a network-centric system, the Future Combat System (FCS), where each component is a complex system in its own right.

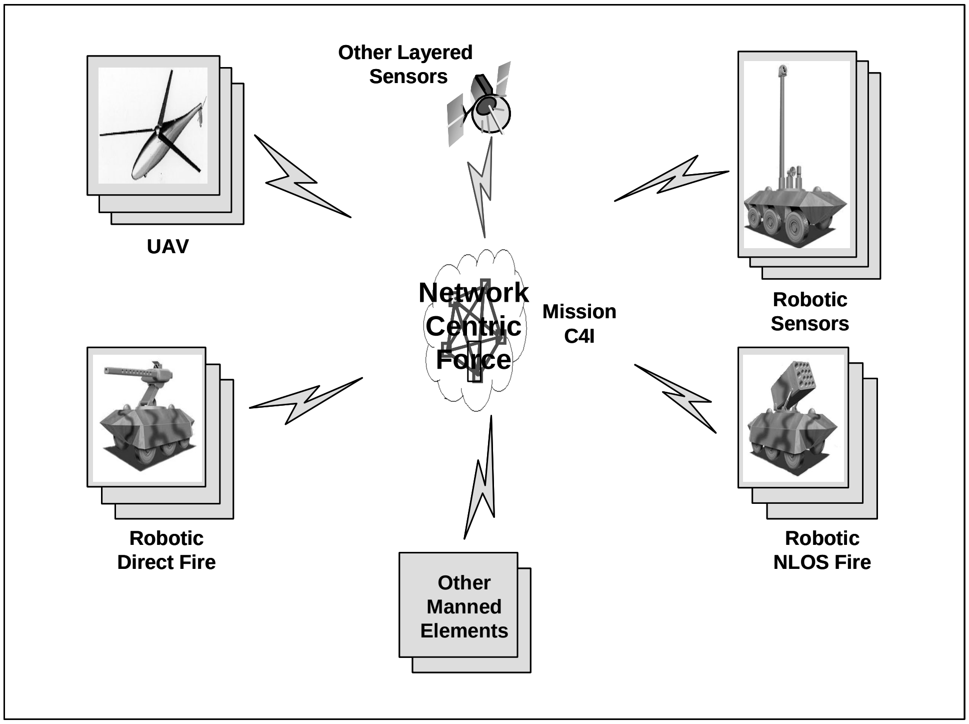

Figure 1: Elements of the Network-Centric Future Combat System

The FCS is highly distributed, contains hundreds of nodes, is operated by thousands of users, conducts complex asynchronous communications and operations, is subject to damage, disruption, and compromise, and undergoes continual upgrade and evolution. Controlling complexity and ensuring survivability are priorities for the development of systems such as FCS.

Similar characteristics are found in commercial network-centric systems. Consider the system-of-systems traversals involved in a gasoline purchase transaction with a credit card depicted in Figure 2. Hundreds of hardware and software components are traversed through many systems in the multiple conversations from gas pump to landline and satellite telecommunications to credit database and back, with many outcomes possible. Each system involved in the network exhibits unique functionality and quality attributes, including reliability, security, and survivability.

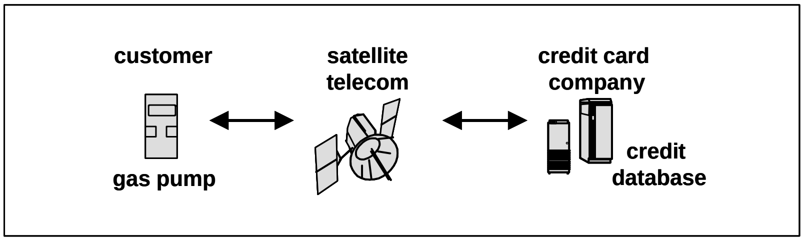

Figure 2:   System-of-Systems Traversals in a Gasoline Purchase Transaction

The burden of unmastered complexity creates difficulties in understanding systems we have and in defining systems we need. It leads to loss of intellectual control when it exceeds human capabilities for reasoning and analysis. Intellectual control means understanding system behavior at all levels in all circumstances of use. It means orderly development and evolution, and no surprises in operation. Intellectual control does not mean the absence of uncertainty, failures, or compromises—they are inevitable—but rather the foresight and capability to deal with them. And it does not require slow and burdensome methods for development. On the contrary, intellectual control enables rapid development with confidence. Without intellectual control, guessing and hoping are the order of the day, and surprises are inevitable.

Complexity barriers can be swept away by the right foundations. When the Normans conquered England in the 11th century, they conducted a census to determine what had been had won. But the results were never added up, despite the obvious interest in such a sum. The census data had been recorded in the Roman system, and the best minds of the day were overwhelmed by the complexity of adding up so many Roman numbers. The representation and reasoning methods were themselves a primary source of complexity. But if the census had been recorded in decimal arithmetic with place notation, any child of the realm could have produced the necessary sums. The right foundations would have made all the difference.

Today, we face complexity and survivability issues on a whole new level in large-scale network systems. Complexity reduction requires solid foundations and engineering methods to maintain intellectual control in specification, design, and operation. Complexity and survivability are closely related. Complexity diminishes survivability by masking potential failures and vulnerabilities, and hiding unforeseen access paths for intrusion. Survivability improvement requires knowing system component dependencies in all usage situations, preparing for component compromises and failures in all situations, and designing system actions for all situations [Ellison et al. 1999c, Mead et al. 2000]. In short, survivability requires intellectual control.

This paper describes the emerging Flow-Service-Quality (FSQ) engineering process that provides foundations and methods for dealing with complexity and survivability in network-centric systems. Section 2 provides an overview of FSQ technologies. Section 3 discusses semantic foundations of Flow Structures, and Section 4 provides illustrations of Flow Structure engineering operations. Section 5 introduces Computational Quality Attributes, and Section 6 discusses Flow Management Architectures. Section 7 summarizes the contributions and implications of FSQ research. Future papers will describe mathematical foundations for Flow Structures and Computational Quality Attributes, and introduce engineering practices for their application.

2 Flow-Service-Quality Engineering

Development of large-scale network systems is essentially a massive integration activity that seeks to reconcile and satisfy user requirements through combinations of COTS and unique components, often within a framework of predefined environments, legacy systems, enabling technologies, and domain architectures. A key issue in modern system development is how to maintain intellectual control over such complex structures and the asynchronous behaviors they produce. In this world of large-scale, asynchronous network systems with dynamic and often uncertain functionality and structure, we ask three questions that deal with engineering methods for complexity reduction and survivability improvement:

1. What are the unifying engineering foundations for system analysis, specification, design, and verification?
2. How should quality attributes such as survivability, reliability, and performance be specified and achieved?
3. What architecture frameworks can simplify system development and operation?

In short, what are the stable and dependable anchors for specification and design that can provide a unified engineering discipline for large-scale network system analysis and development? Our research is developing new approaches to answer these questions. The following concepts help to structure our research thinking:

1. Flow Structures

   User task flows and their refinements into system service uses can provide unifying engineering foundations for analysis, specification, design, and verification of functionality and quality attributes.
2. Computational Quality Attributes

   Quality attributes can be associated with both flows and the system services they invoke, and specified as dynamic functional properties to be computed, rather than as static, a priori predictions of uncertain utility in real-time system operations.
3. Flow Management Architectures

   Flow Structures and Computational Quality Attributes support canonical architecture frameworks that manage flows, network services, and their quality attributes in execution.

Flow Structures are compositions of system services that carry out user tasks to accomplish enterprise missions. They employ unique semantics to preserve important deterministic properties for precise human understanding and analysis, despite the underlying asynchronism and unpredictability of network behavior. Flow Structures take into account unpredictable events and outcomes that can impact mission survivability. Computational Quality Attributes of flows and the services they invoke can be dynamically managed in execution. Thus, the first-class concepts of flow, service, and quality form the basis for the emerging discipline of Flow-Service-Quality (FSQ) engineering [Hevner et al. 2001, Hevner et al. 2002]. A persistent problem in development and management of large-scale network-centric systems has been the lack of unified, scale-free engineering foundations for intellectual control in management, acquisition, analysis, development, evolution, and operation. FSQ engineering addresses that problem through theoretical foundations and practical engineering methods to represent, analyze, develop, and dynamically manage system flows and their quality attributes as essential and primary artifacts of network system development.

Distributed information systems are usefully viewed as networks of asynchronously communicating components that provide system services whose functions can be combined in various patterns to satisfy enterprise mission requirements. System services include all the functional capabilities of a network system, from communication protocols and operating systems to databases and applications. The sequencing of system services in user task flows can be mapped into compositions of network hardware, software, and human components that provide the services. These compositions are end-to-end traces that define slices of network architectures whose net effect is to carry out operations that satisfy user requirements. Figure 3 depicts refinement of user task flows based on mission objectives into uses of system architecture components.

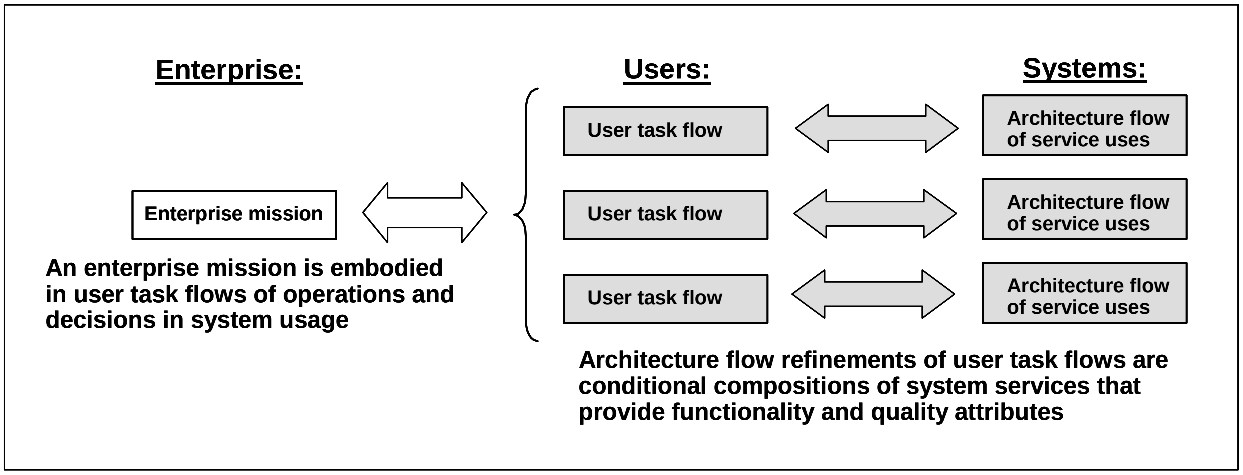
Enterprise:
Users:
Systems:
Enterprise mission
User task flow
User task flow
User task flow
Architecture flow of service uses
Architecture flow of service uses
Architecture flow of service uses
An enterprise mission is embodied in user task flows of operations and decisions in system usage
Architecture flow refinements of user task flows are conditional compositions of system services that provide functionality and quality attributes

Figure 3:   Refinement into User Task Flows into System Service Uses

Large-scale systems support many simultaneous users in many roles with many possible task flows, and particular system services may appear over and over in their definitions. In fact, a principal design objective in large-scale systems is coordination and synchronization of multiple uses of particular services incorporated into flows. In dynamic networks with constantly varying function and usage, flows and their corresponding architecture traces of system services [Parnas and Wang 1994] act as stable foundations for functional and non-functional, that is, quality attribute, specification and design. In execution, services invoked by flows can experience a blizzard of asynchronous usage interleavings that defies human understanding. A key property of Flow Structures is a semantic foundation that permits flows to exhibit deterministic properties for straightforward human understanding and analysis, despite the underlying asynchronous behavior. Thus, flows can be represented as simple procedural structures composed of nested and sequenced service invocations and local computations expressed in terms of ordinary sequence, alternation, iteration, and concurrent structures. Such structures define an algebra of component composition with desirable properties. For example, Flow Structures preserve effective reasoning methods of composition and referentially transparent refinement, abstraction, and verification for human understanding. Flows can be expressed in virtually any language using Flow Structure primitives to specify user task uses of system services in precise terms. Services invoked by flows can be refined into flows invoking other services, etc., in a recursive design process that employs identical structures and engineering methods at all levels of refinement. Flow Structures are superficially related to workflow methods [Leymann and Roller 2000], but define rigorous scale-free foundations for large-scale system analysis, development, and operation.

Flows can be organized into related FlowSets associated with particular components and network partitions. Transitivity analysis of flows can reveal often-unforeseen dependencies. Flows define required levels of quality attributes for themselves, as well as for execution of the services they reference. FSQ engineering operations for existing and new systems are depicted in Figure 4. For new systems, flow specification begins with user tasks that support enterprise mission objectives, thereby ensuring a user-centric approach to design and development. In particular, flows are vehicles for definition of required quality attributes, such as reliability and survivability. Some flows require higher levels of reliability and survivability than others, and flow-specific definition of attribute requirements permits informed cost/benefit tradeoffs in system design. For existing systems, flows of mission-critical operations can be extracted and analyzed to reveal unforeseen dependencies and single points of failure. Such analysis permits identification and development of survivability improvements. It is important to note that flows can define both legitimate and illegitimate use. Intruder usage of systems can be expressed in flows to reveal compromisable components and help define security and survivability improvements [Mead et al. 2000, Moore et al. 2001].

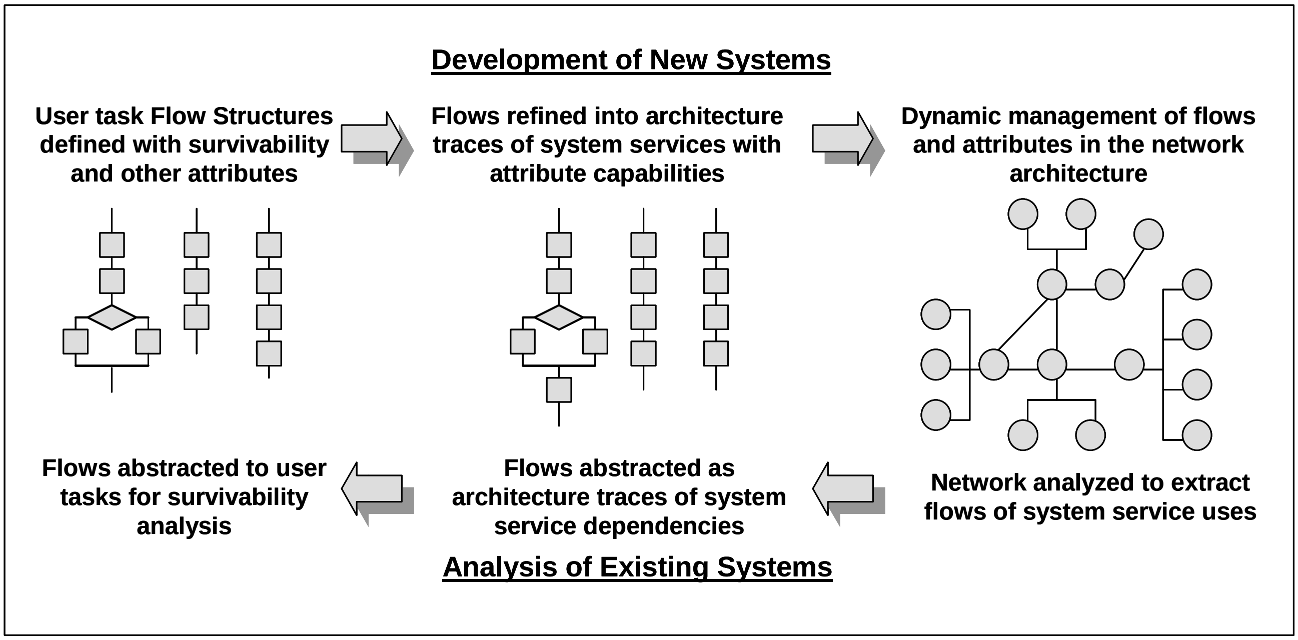
Development of New Systems
User task Flow Structures defined with survivability and other attributes
Flows refined into architecture traces of system services with attribute capabilities
Dynamic management of flows and attributes in the network architecture
Flows abstracted to user tasks for survivability analysis
Flows abstracted as architecture traces of system service dependencies
Network analyzed to extract flows of system service uses
Analysis of Existing Systems

Figure 4: FSQ Engineering Operations for New and Existing Systems

In terms of intuitive understanding, flows can be thought of as deterministic hierarchies of service uses superimposed for execution with other flows on a large-scale asynchronous network. This concept is depicted in Figure 5 for a flow in the Future Combat System example. Flows define a structured use of network capabilities through definition of communications among, and compositions of, network system services.

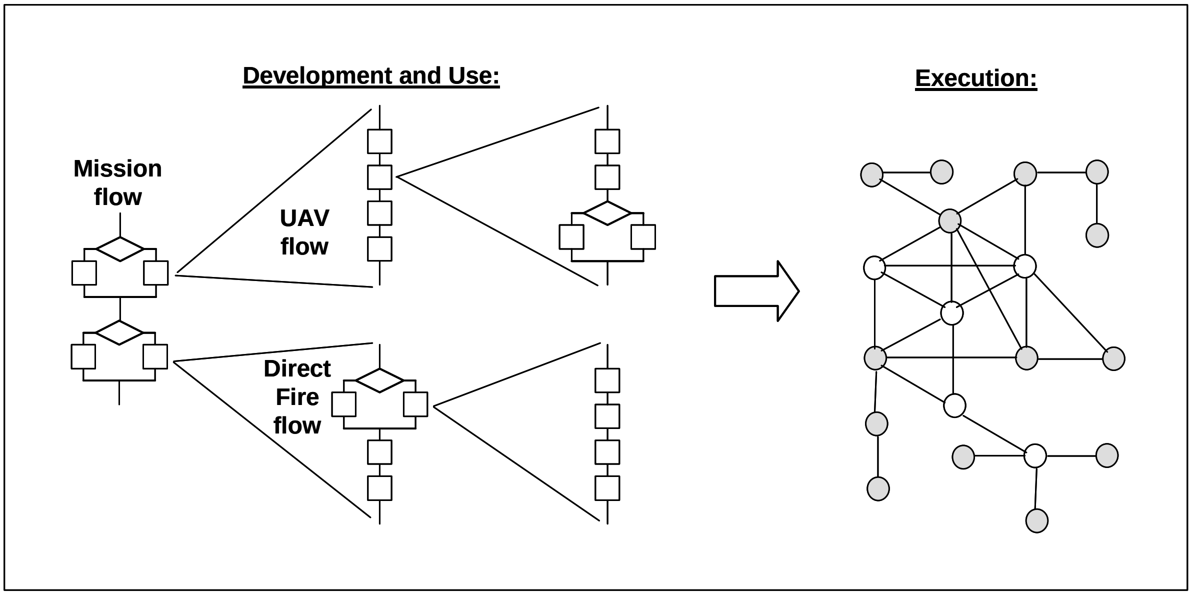

Figure 5: Superimposing a Deterministic Flow on an Asynchronous Network

In FSQ engineering, quality attributes such as availability, reliability, and survivability are defined as computational functions, and are associated with both flows and services. Substantial effort has been devoted in the past to development of a priori characterizations of quality attributes. Rather than focusing on descriptive methods of limited value for dynamic networks, we adopt an alternate approach and ask how such attributes can be defined, computed, and acted upon as dynamic characteristics of system operation. That is, we wish to define quality attributes as functions to be computed, rather than as static descriptions of capabilities to be achieved. While such functions rely on what can be computed and may differ thereby from traditional views of quality attributes, they can permit new approaches to attribute analysis, design, metrics, and operational evaluation. A key aspect of the computational approach is the ability to associate quality attributes with specific flows rather than with entire systems, thereby permitting differentiation among attribute capabilities based on mission criticality in survivability engineering. Some quality attributes, such as availability and reliability, are readily quantified for computational analysis. Others, such as security and survivability will be more difficult. Nevertheless, we believe the effort must be made, and that new approaches and insights will result.

In a world of Flow Structures and Computational Quality Attributes, it is natural to consider system architecture frameworks based on dynamic flow and quality attribute management [Sikora and Shaw 1998, Haeckel 1999, Sullivan et al. 1999]. A primary control task in large-scale systems is managing the sequencing of system services in real time to satisfy flow specifications. FSQ concepts suggest standard, subject-matter-independent Flow Management Architecture (FMA) frameworks for dynamic flow and attribute management in execution. Such frameworks could reconcile flow requirements with service availabilities, and implement operational management strategies based on dynamic network and service capabilities and workloads. FMA frameworks embody the concept of an FSQ Manager, centralized or decentralized within the architecture of a system, which provides such flow management. In particular, an FSQ Manager could provide dynamic quality attribute evaluation. For example, survivability management could include a variety of strategies such as rerouted communication paths, resource substitutions, state reconstruction, alternate provisioning, and system reinitialization and reconfiguration. An FSQ Manager could be designed and instantiated in a variety of forms and technologies, depending on user requirements, network configuration, and the operational environment.

FSQ engineering can reduce complexity and add clarity to network system development. Flow Structure specifications of enterprise tasks can be designed and verified with full human understanding at various levels of abstraction in a seamless process from user task flows down to architecture components. Flow Structures prescribe logical network connections and operations, define compositionality among nodes and services, and support both centralized and distributed control. The specification of network system behavior and logical connectivity is defined by the set of flows of its service uses. The specification of each service in a network system incorporates all its uses in all the flows in which it appears. Flow Management Architecture frameworks provide systematic templates for managing flow instantiations and reconciling Computational Quality Attributes.

3 Flow Structure Semantics

3.1 The Semantic Model

In large-scale network systems, flows can engage in extensive traversals of network nodes and communication links, where the behavior of invoked services cannot always be known and predicted. In this environment a variety of Uncertainty Factors must be managed, including:

1. Unpredictable function

   A service may be provided by COTS or External Service Provider (ESP) components of unpredictable function and reliability that may not perform expected operations every time or any time it is invoked.

2. Compromised function

   A service may have been compromised or disrupted by an intrusion or physical attack and may not be able to perform its function correctly or at all.

3. High-risk function

   A service may not provide adequate levels of quality attributes (Quality of Service) required by a flow.

4. Modified function

   A service may be modified or replaced as part of routine maintenance, error correction, or system upgrade, with intentional or inadvertent modification of its function.

5. Asynchronous function

   A service may be used simultaneously and asynchronously by other flows, and thus produce results dependent on unpredictable history of use, both legitimate and illegitimate.

These factors are pervasive behavioral realities of large-scale, network-centric systems [Schneider 1999]. Dealing with them is an enterprise risk management problem with potentially serious consequences. It is important to detect when they have occurred and take appropriate actions to continue operation in the environments they have created. For mission-critical flows, these actions must ensure survivability no matter what adverse environments are encountered [Mead et al. 2000]. In today's world, it is imprudent from a risk management perspective to fail to fully address the Uncertainty Factors at all levels of enterprise and system operation.

The mathematical semantics of Flow Structures are defined to support development and verification of flows for such uncertain environments as a standard engineering practice. To allow for unpredictable behavior of services, flow semantics permit specification of only the processing that a flow itself performs, and not the processing of the services it invokes. Flow Structure engineering requires definition of appropriate actions by a flow for all possible responses of key services, both desired and undesired. Thus, if the behavior of invoked services changes for any reason, the specification and verification of the invoking flow need not change. This approach accommodates the realities of today's network systems and offers important advantages. It requires for mission survivability that the Uncertainty Factors be dealt with explicitly in design, thereby addressing important aspects of enterprise risk management. It permits flows and reasoning about them to be localized yet complete. And it permits flows to be defined by simple deterministic structures despite the underlying asynchronous behavior of their constituent services. These deterministic structures can be refined, abstracted, and verified using straightforward compositional methods for human understanding and intellectual control.

It turns out that these objectives require extension of the traditional functional semantics model. The FSQ semantic model is based on the well-known concept of services as rules for mathematical functions (or relations if flows include concurrent operations), that is, mappings from domains (inputs, stimuli) to ranges (outputs, responses) [Linger et al. 1979, Mills et al. 1986, Prowell et al. 1999, Hoffman and Weiss 2001, Mills and Linger 2002]. The key extension required to deal systematically with the Uncertainty Factors is to make the histories of service invocations themselves part of the specified behavior of flows. Mathematically, this is achieved by including the invocation stimulus history (ISH) of every service in the range of the function that represents the specification of a flow. In addition, because subsequent flow processing can depend on the responses from these invocations, the invocation response history (IRH) must be part of the domain of the mathematical function that represents the specification of a flow. The diagram of Figure 6 illustrates these semantics for a flow F invoking a service A.

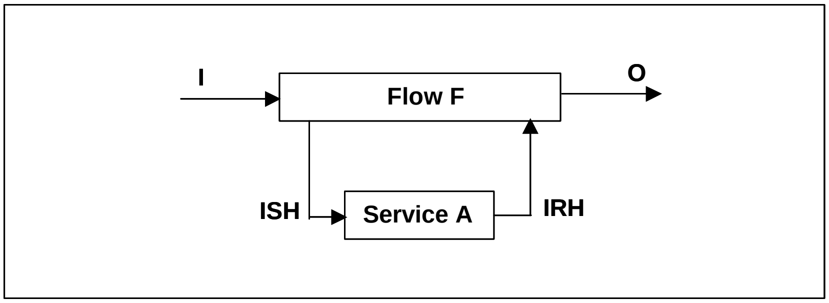

Figure 6: Elements of Flow-Service Semantics

I is the set of possible inputs to flow F, and O is the set of possible outputs from flow F. Thus, the semantics of F can be given by a mathematical function f with domain I x IRH and range O x ISH. It is this counterintuitive inclusion of service responses in the domain of F and service stimuli in the range of F that allows flows to deal with the Uncertainty Factors. In particular, IRH represents the range of possible service responses, and thus embodies the Uncertainty Factor possibilities that should be dealt with in flow design. Dealing with the Uncertainty Factors requires assessing and acting upon all possible responses, desired and undesired, that service invocations can produce. Of course, no semantics can force such informed design, they can only illuminate the desirability of doing so.

In this semantic model, the specification of flow F is not required to account for the behaviors that result due to invocation of service A. Rather, it simply defines the invocation of service A with certain parameters, and how the response from that invocation affects subsequent processing of F. This means, for example, that any lower-level services invoked by service A need not be part of the ISH and IRH of flow F. If this were not the case, the specification of F would change if service A was modified, for example, to invoke different lower-level services. This approach differs from traditional functional semantics, where the specification of F would be required to include the full effects of all lower-level service invocations by service A as a part of its functional specification.

This approach to specification is key to maintaining intellectual control over flow specification and design. As noted, deterministic flows that invoke non-deterministic, asynchronous services can be modeled by deterministic mathematical functions, making human reasoning and analysis much simpler. Alternately, if the behavior of flows were non-deterministic, then the flows themselves would become far more complicated, and their semantics would need to be expressed as a mathematical relation from domain I x IRH to range O x ISH. This complex situation is avoided by FSQ semantics.

The flow semantics described above are particularly suited to the common situation where service A already exists on a network, or is provided by COTS or ESP components with complex and possibly unknown functions. In cases where service A is new and must be designed as part of the implementation of flow F, these flow semantics can be combined with traditional design and verification methods such as those found in object-based box structures [Mills et al. 1986] to support reasoning about the combined behavior of the system consisting of F and A together. In this way, the desired behavior of F and A can be used to guide the construction of A. In particular, box structures provide history-, state-, and procedure-oriented representations of flows and services, and methods for their abstraction, refinement, and verification.

Flow Structure concepts and techniques apply to network flows written in almost any imperative language, including C++ and Java, provided the language includes the basis set of control structures, namely, sequence, alternation, and iteration. Specializations and extensions of these structures are valuable as well. Service invocations in these languages are method calls on objects. An abstract Flow Structure Language (FSL) can be defined that captures the essence of FSQ concepts independently of specific implementation language syntax. To deal with concurrency within a flow itself, a concurrent structure is included in FSL as well. Because specifics of language data type and declaration syntax do not affect the applicability of FSQ, these features can be de-emphasized in FSL. Figure 7 illustrates typical FSL control structures.

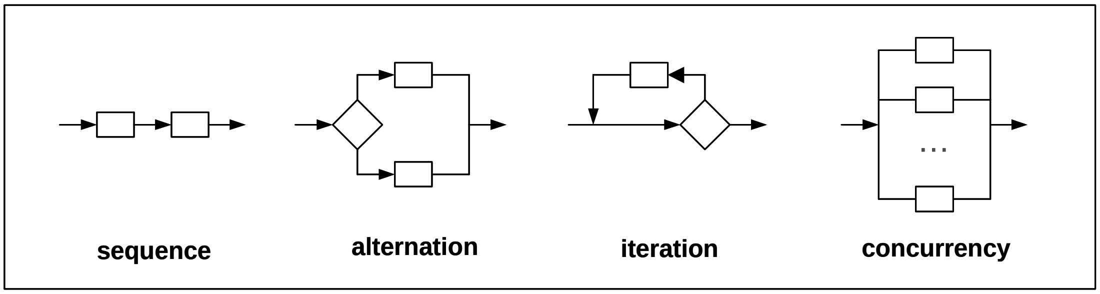

Figure 7: Typical FSL Control Structures

The overall behavior of an individual flow is as follows: A flow is invoked with values assigned to its input parameters, and at the termination of flow execution, the final values of output parameters are returned to the invoker of the flow, whether a human user or another flow. A flow can define local, non-persistent data to store intermediate values produced by flow computations. Finally, a flow can invoke services to accomplish various network or even local activities, including storing, accessing, and modifying persistent data. In designing flows, this means that the persistent state required by a flow should be encapsulated in services.

In addition to sequence, alternation, iteration, and concurrent structures, FSL contains a “use” statement to invoke services. The statement incorporates post-fix predicates to evaluate and act upon designer-defined equivalence classes on the set of all possible responses, both desirable and undesirable. This partitioning and analysis of response equivalence classes addresses the requirement to deal with the Uncertainty Factors characteristic of network behavior, and the survivability implications they impose. Flow designers select key services for such response analysis. The general syntax of the use statement is as follows:

```
use <service>.<method>(<parameters>)
   response <status_variable> is
       <enumerated_value_1> when <expression>
     | <enumerated_value_2> when <expression>
     | ...
     | <enumerated_value_n> when <expression>;
```

For example, the following use statement illustrates invocation of an airline reservation database to reserve space on a flight:

```
use Airline.reserve(customer, flight, date, result, seat)
   response status is
```

```
  NOTRESERVED when result = false
| RESERVEDNOSEAT when (result = true) and (seat = "")
| RESERVEDWITHSEAT when (result = true) and (seat != "");
```

In addition to such explicitly enumerated equivalence classes on the response, parameters based on network and component status (e.g., NOTCONNECTED, NORESPONSE) could be evaluated as well. Such evaluations are important in assessing and acting upon dynamic survivability properties of a network.

3.2 FSQ Theorems

A number of theorems capture and explore the fundamentals of FSQ semantics. Example theorems are described below. Proofs are beyond the scope of this paper and can be found in [Pleszkoch et al. 2002]:

1. *Flow Structure Theorem*

   Given any graph representing a flow there exists an equivalent flow that can be implemented using only composition, alternation, and iteration control structures.

   Engineering Implications:

   This theorem guarantees that composition, alternation, and iteration control structures are sufficient to implement any flow. Thus, flow developers need not use unstructured logic or arbitrary branches.

2. *Abstraction/Refinement Theorem*

   Two flows F and G are equivalent in any network environment if and only if they have identical flow specifications.

   Engineering Implications:

   This theorem is the basic justification that FSQ mathematical semantics are correct. It consists of two parts, necessity and sufficiency. Sufficiency states that any two flows that have the same mathematical specification can be interchanged without affecting the results of any larger network environment. Necessity states that for any two flows that have different mathematical specifications, there exists a larger network environment that will produce different results if they are interchanged. In essence, this theorem says that everything that is important about the behavior of a flow from the point of view of the external network is contained in its specification.

3. *Flow Verification Theorem*

   The basis set of single-entry, single-exit control structures that comprise flow designs can be verified by evaluating the function equations shown below for representative structures. Each equation is followed by a statement of a Correctness Question that articulates the verification conditions. Verification requirements for similar control structures are easily derived. In order to support function composition of flow specifications in verification operations, their domain and range must be extended to a common superset. In the case of a flow specification [F] = f : I x IRH -> O x ISH, f is first extended to the function f' : S x IRH -> S x ISH, where S is the functional verification state space corresponding to the input and output parameters and local variables of the flow F. Next, f' is extended to the function f'' : S x IRH x ISH -> S x IRH x ISH, given by f''(s,rh,sh) = (s', rh', sh'), where (s', sh') = f'(s,rh), and rh' is equal to rh with the first n elements removed, where n is the length of sh'. By making this extension, the semantics of an entire flow can be calculated by a functional verification trace table with an extra column for the response history variable rh. Thus, for a given flow specification f, services g and h, predicate p, and for all possible arguments to f, control structures can be verified as follows: [Mills 1988, Prowell et al. 1999]:

Composition structure:

f = g; h   Does g followed by h do f?

Alternation structure:

p --> f = g | ~p --> f = h   Whenever p is true, does g do f? and

Whenever p is false, does h do f?

Iteration structure:

termination $\wedge$ Does the iteration terminate? and

p --> f = g o f $\wedge$ Whenever p is true does g followed by f do f? and

~p --> f = null   Whenever p is false, does doing nothing do f?

Engineering implications:

The Flow Verification theorem reduces verification to a finite and complete process despite the fact that flows can contain a virtually infinite number of paths. Verification can be carried out using Trace Tables [Prowell et al. 1999] as a formal analysis and documentation process for critical system parts, or applied in team reviews with greater speed and little loss of precision.

4. *Flow Implementation Theorem*

For each computable function f from I x IRH -> O x ISH that satisfies the following condition, there exists a flow F such that [F] = f.

Condition: For every $n \geq 0$, there exists a function f_n : I x IR^n -> O x $IS^{(n+1)}$, where IR^n is the first n elements of IRH, and $IS^{(n+1)}$ is the first (n+1) elements of ISH, such that the following diagram commutes:

$$\begin{tikzcd}
I \times IRH \arrow[r, "f"] \arrow[d, "PI_n"] & O \times ISH \arrow[d, "PO_n"] \\
I \times IR^n \arrow[r, "f_n"] & O \times IS^{(n+1)}
\end{tikzcd}$$

where PI_n : I x IRH -> I x IR^n is given by PI_n(i, rh) = (i, first n elements of rh), and PO_n : O x ISH -> O x $IS^{(n+1)}$ is given by PO_n(ø, sh) = (null, first n+1 elements of sh) if sh has at least n+1 elements, and by PO_n(ø, sh) = (ø, sh) if sh has n or fewer elements.

Engineering Implications:

Every flow F has a function [F], but it turns out that not every function f corresponds to a flow. This theorem defines which functions do in fact correspond to flows. Informally, the condition states that a function cannot require a flow to predict the future, by making decisions based on responses from services before those services are actually invoked. The theorem states that any flow semantics that meets this condition can actually be implemented. Thus, this theorem is important to take into account when designing flows top-down.

5. *System Testing Theorem*

Let D_I be a usage distribution on the input of a flow F, and let D_R be a usage distribution on the responses to the external service calls made by F. Then the usage distribution D_S on the stimuli of the external service calls made by F can be calculated from D_I, D_R, and [F].

Engineering Implications:

Systems of any size exhibit a virtually infinite population of possible executions. Therefore, all testing is sampling, and the only real question is how to draw the sample of test cases to be executed. If a sample is representative of actual field usage, scientifically valid predictions based on the sample test results can be made for the population of all executions not tested, which, of course, users will encounter in field usage. Such modern usage-based statistical testing supports effective test management and risk reduction processes. Flow specifications and effective system testing processes are closely related. Flows define how systems are used. Given usage frequencies for flows that define when and how they are used, it is possible to predict the usage of their services. Such predictions can populate the probability distributions of usage models that are employed to generate test cases (samples of the execution population) statistically faithful to anticipated usage. Such an approach to testing permits valid estimates of system performance in field use, and thus guides test management and product release decisions.

Other theorems provide additional foundations for FSQ engineering operations, including transitive analysis of flow dependencies and derivation of logical system architectures from flows.

4 Flow Structure Engineering Operations

Flow Structures support many engineering operations in network system analysis and development. Some representative operations are briefly described below:

4.1 Flow Engineering for Uncertainty Factors

Uncertainty Factor engineering requires that flows address all possible responses (IRH) from critical services. This engineering process requires definition of post-fix predicates on responses to determine and design appropriate actions. Responses can be organized by designers into subject-matter-dependent equivalence classes of interest. These equivalence classes are the subject matter of risk management and mission continuity in survivability engineering. It is up to designers to select those critical service invocations that should undergo response analysis.

Figure 8 illustrates use of post-fix predicates in a notional fragment of an air traffic control flow. A controller using the flow is expecting to identify an aircraft (use a/c ident) and obtain its position fix (use a/c position fix). Three post-fix predicates follow the invocation of the a/c ident service. These predicates parse the service response into equivalence classes dealing with whether any response was received, whether it was an aircraft identification, and whether it was a valid aircraft identification. In this small example, each case of negative evaluation notifies the controller through the controller interface service. But consider design-time issues and discussions in development of a system to support this mission-critical flow. Failure of the a/c ident service is a very serious matter impacting completion of the flow and thereby the safety of aircraft. Transitivity analysis of other flows upon which a/c ident depends may reveal a whole series of cascade failure possibilities that must be dealt with to ensure survivability of this critical service. Such analysis may result in major changes to the proposed system architecture. In any event, it should become clear that notification of the controller is an insufficient action for this serious problem, and that this flow must be redesigned. Note in this discussion that the semantics of Flow Structures permit designers to define and act upon response equivalence classes that encompass the Uncertainty Factors. This supports enterprise risk management, which requires analysis of all possible outcomes, and survivability management, which requires actions for all possible outcomes.

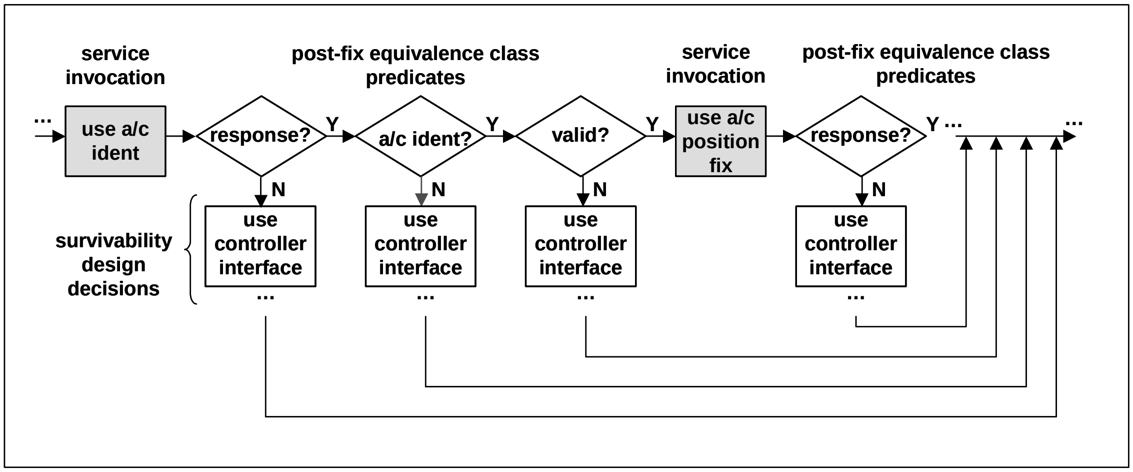

Figure 8: Post-Fix Service Response Evaluation for Survivability Analysis and Risk Management

4.2 Flow Abstraction and Refinement

The single-entry, single-exit basis set of control structures that comprise FSL can be composed and nested to create Flow Structures of any size and complexity. As noted above, flows and their constituent control structures implement mathematical functions or relations, that is, mappings from domains to ranges. These functions define data to be transformed from initial to final values, and can be included in flows as comments attached to their control structure refinements. They can be defined in a spectrum of forms, ranging from natural language to more mathematics-based notations.

An algebra of functions permits precise abstraction and refinement in the substitution of function definitions for their refinements (abstraction to determine what flows actually do) or substitutions of designs for their function definitions (refinement to implement what flows are intended to do). Figure 9 depicts these operations in abstract form. Design abstraction occurs in moving left to right, where the overall function of the flow is abstracted in three steps to function C. Design refinement occurs in moving right to left, where the full elaboration of function C is achieved in three steps.

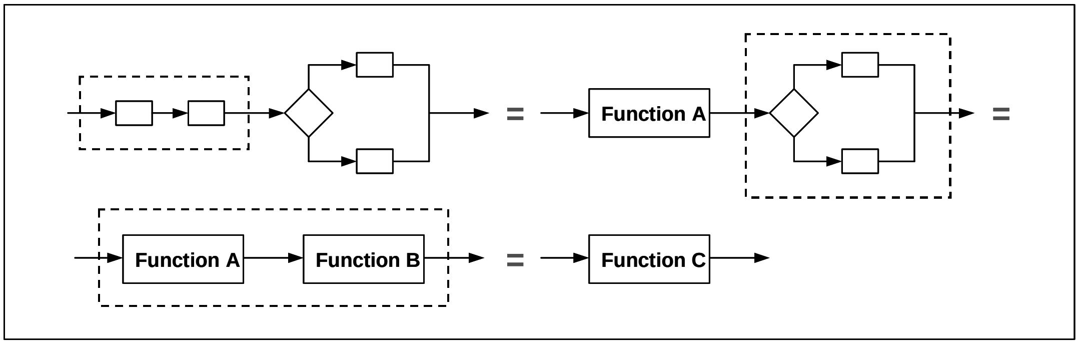

Figure 9: Algebraic Operations in Flow Structure Analysis and Design

Figure 10 provides a miniature notional illustration of flow refinement and abstraction based on the example of a gasoline purchase transaction. This informal depiction shows the flow at a mission level of abstraction that is refined into a specification of principal operations and their composition. The first specified operation is further refined into a high-level design. Abstraction reverses this process ending in the mission description of the flow [Hausler et al. 1990, Pleszkoch et al. 1990].

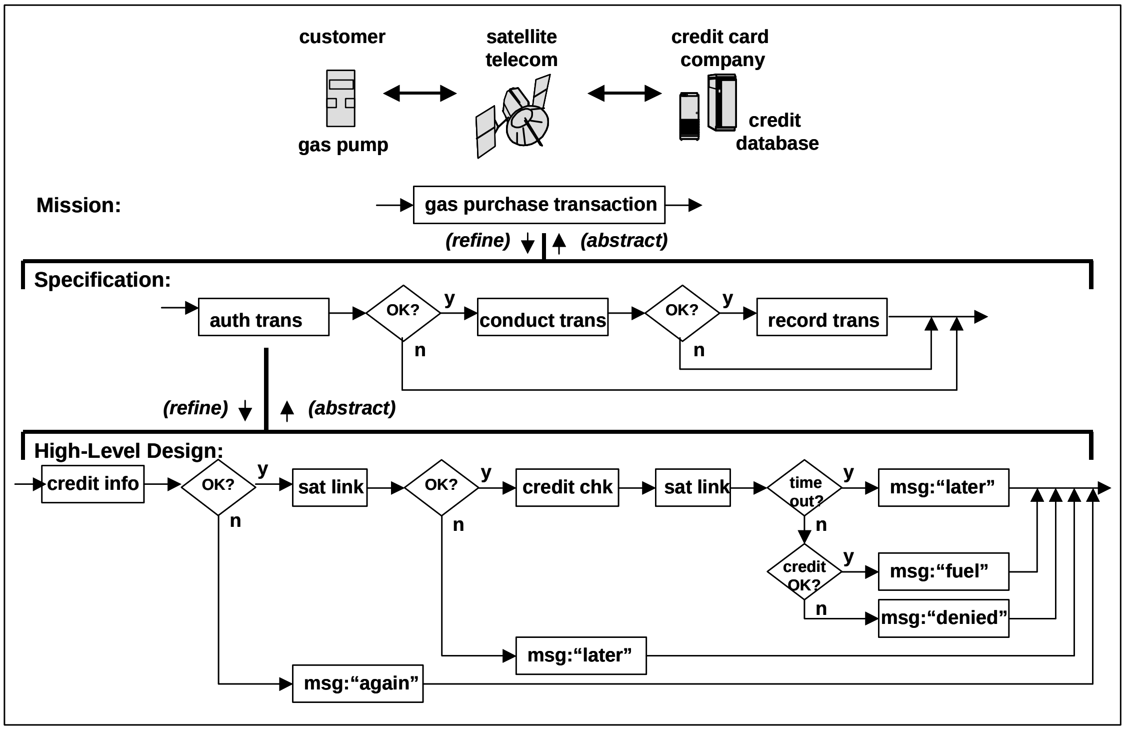

Figure 10: Flow Abstraction and Refinement in a Transaction-Based System

Informal methods of abstraction and refinement are depicted in Figure 10. However, the semantics of Flow Structures permits these operations to be carried out with precision, to support predictable assembly of components in composition operations, and to allow verification of components with respect to their specifications.

4.3 Flow Verification

The Flow Verification theorem provides the basis for an effective team-oriented assurance process. Flows annotated with function definitions of their constituent control structures can be verified in structured team reviews that ask and answer the Correctness Questions and require unanimous agreement on each question.

Flows may contain a virtually infinite number of possible execution paths impossible to verify on an individual basis. However, they are composed of a finite number of control structures, and the Flow Verification Theorem permits every control structure to be verified in a finite number of steps [Mills et al. 1986, Prowell et al. 1999]. One step is required for sequence verification (function composition), two steps for alternation verification (true and false case analysis), and three steps for iteration verification (termination proof, plus true and false case analysis and function composition). Thus, verification is reduced to a finite process amenable to team operations. Such team verifications are cost effective in detecting problems and errors early in development for correction at lowest cost. Figure 11 illustrates the flow verification process. As shown on the left, a miniature flow composed of a sequence followed by an alternation is to be verified. The sequence, the alternation, and the composition of these two structures must each be verified to be correct with respect to their intended functions. Errors are discovered in the center of the figure based on application of the Correctness Questions, and are shown corrected on the right.

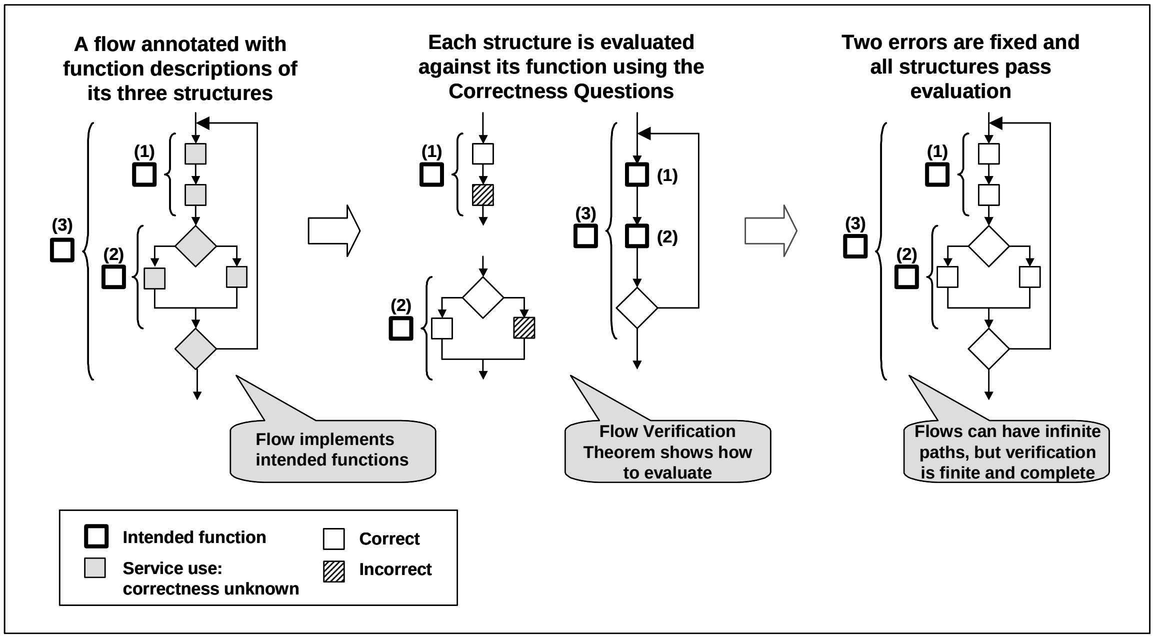
A flow annotated with function descriptions of its three structures
(1)
(2)
(3)
Flow implements intended functions
Each structure is evaluated against its function using the Correctness Questions
(1)
(2)
(1)
(2)
(3)
Flow Verification Theorem shows how to evaluate
Two errors are fixed and all structures pass evaluation
(1)
(2)
(3)
Flows can have infinite paths, but verification is finite and complete
Intended function
Service use: correctness unknown
Correct
Incorrect

Figure 11: The Correctness Evaluation Process Based on the Flow Verification Theorem

Each verification of a control structure against its intended function requires only local reasoning, and the verifications can be carried out in any order. When more precision is required for critical flows, verification can be accomplished with trace tables to document reasoning and analysis. When a service is provided by a COTS component with complex functionality, it is necessary only to verify the use actually made of the component. This verification will include the equivalence class evaluations that account for all possible outcomes of the COTS invocation.

4.4 Flow Transitivity Analysis

Flows may invoke services composed of flows that may likewise invoke services composed of flows at a lower level, etc. Thus, a primary flow may depend on completion of many other flows, possibly distributed across many components in a large-scale network. Transitivity analysis of flows can reveal such dependencies for analysis of survivability and other quality attributes.

Figure 12 depicts the beginning of such an analysis for the Future Combat System. The primary mission control flow in the center of the figure named Target Attack invokes a sensor data service and its flow provided by a UAV node, and a fire control service and its flow provided by a robotic direct fire node. These flows in turn invoke other services local to their nodes. So the first step in transitivity analysis is to determine what services and their flows are directly invoked in satisfying a primary flow. In addition, every flow can exhibit both desired and undesired outcomes as defined by equivalence classes on service responses. Clearly, what is intended in a flow invocation is a desired outcome. It is often the case that desired outcomes depend on successful completion of flows not directly invoked. For example, to be operational at all, a UAV component must have undergone successful completion of flows dealing with inventory, training, maintenance, weather analysis, mission definition, fueling, launch, and flight, to name a few, in order for a desired outcome to be obtained when the sensor data service is invoked. This chain of dependencies represents the complete set of UAV flows that supports the mission of the primary target attack flow. Such analysis can reveal surprising and often unforeseen dependencies that must be dealt with to ensure survivability of mission flows. Flows important to enterprise objectives may be vulnerable to poorly designed and implemented flows that would otherwise be hardly noticed given their physical or temporal separation from primary mission flows.

Mission-oriented flows compose and integrate local network services into coherent capabilities that are the reason for existence of the network system in the first place. As such, they serve as overarching specifications for network-level capabilities and the local services that support them.

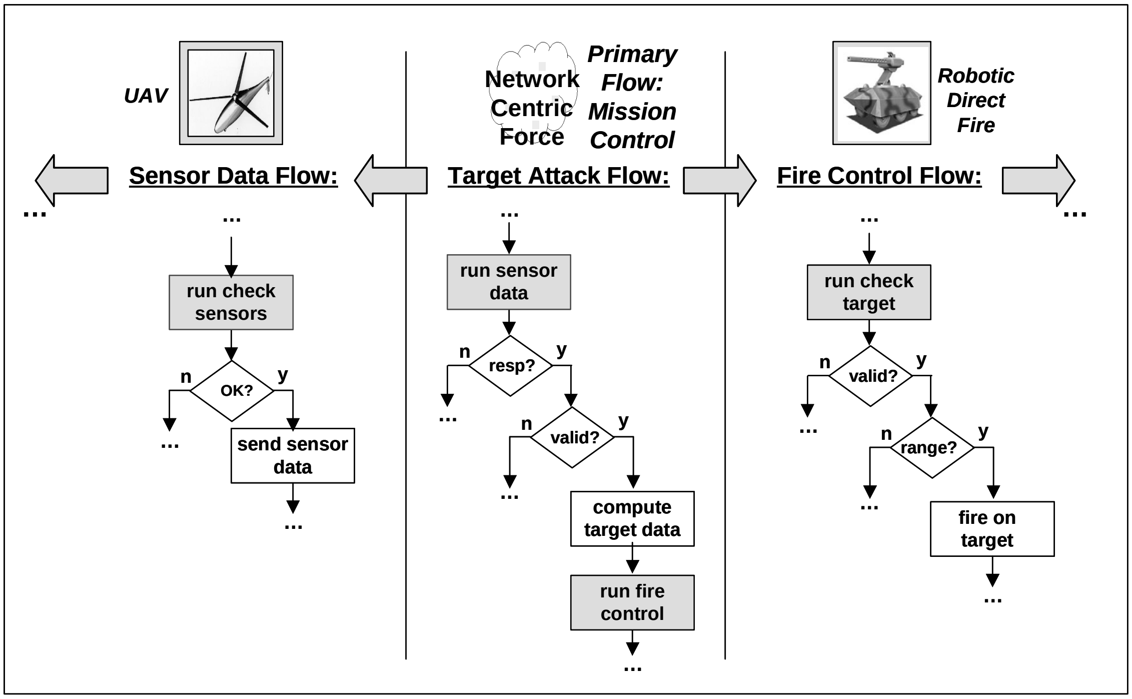

Figure 12: Transitivity Analysis of Flow Dependencies

4.5 FlowSets in Large-Scale Systems

Large-scale systems typically contain components that are complex systems in their own right. As such, these components support services and flows that manage and use their own capabilities, as well as participate in inter-component, network-spanning flows. It is often useful to group services and flows that support particular components or functions. Such groupings are termed FlowSets.

For example, consider FlowSets associated with the Future Combat System as depicted in Figure 13. Each sensor and weapons-delivery node is a full-scale system exhibiting complex functionality, with its own acquisition, development, and evolution life cycles, and unique operational capabilities, vulnerabilities, and constraints. These nodes are designed to support local FlowSets dealing with achieving and maintaining operational capabilities. But they are also designed to participate in network-centric operations to accomplish overarching mission objectives. In the illustration at the center of Figure 13, a FlowSet is defined that supports mission operations by traversing and integrating the capabilities of network components into coherent overall capabilities. For example, sensor integration flows in the FlowSet combine the outputs of layered sensors into a comprehensive assessment of attack opportunities. Fire integration flows select and coordinate attack elements, and damage assessment flows integrate sensor reports of attack outcomes. Such network-centric FlowSets combine the capabilities of individual systems to achieve mission objectives.

So the task of network-centric system development is to define FlowSets for individual systems that seamlessly support mission FlowSets. And mission FlowSets must be designed to integrate these systems into coherent capabilities. In this role, mission FlowSets are the primary specification of network capabilities, and systems within the network must be designed to support them.

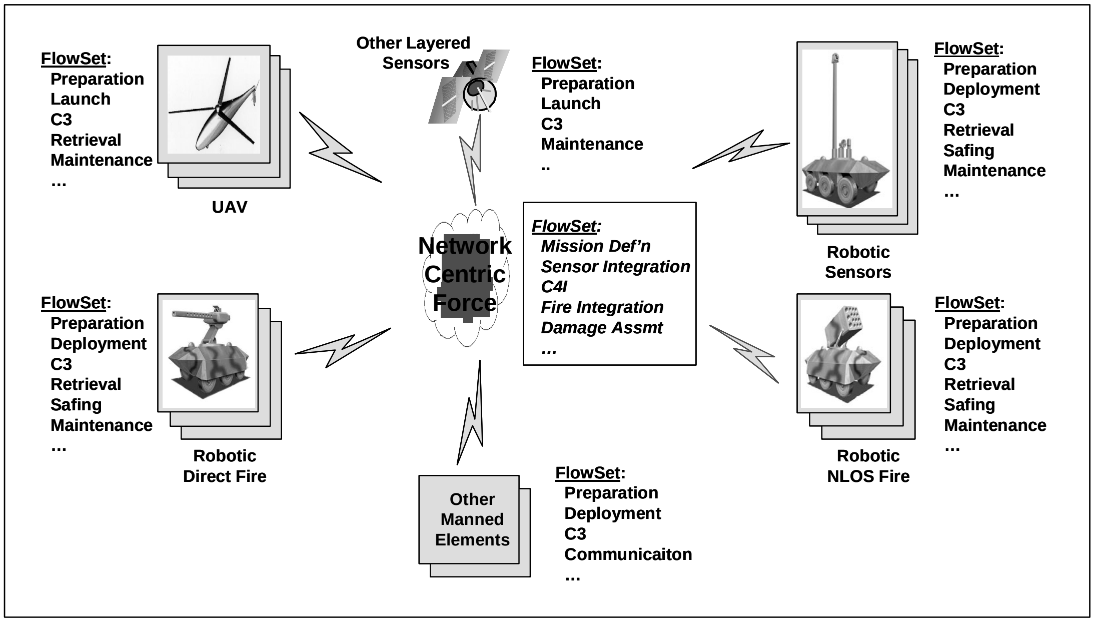

Figure 13: Flow-Sets for the Future Combat System

4.6 Flow Security and Survivability Analysis

Flows represent end-to-end processing, that is, traversals from a user (e.g. person or flow) through a network and back to the user. Many systems and domains may be visited in such traversals, each with its own security and survivability policies and capabilities. This situation is depicted in Figure 14 for the gas purchase transaction example. It is this end-to-end transaction that must be secure and survivable. Quality attributes for these properties can be defined for the overall flow, as benchmark requirements that all traversed systems and domains must provide. Security and survivability can be analyzed with respect to flow propagation, and are only as good as the least capable components. Some flows must be more secure and survivable than others, and service vendors and providers may offer a spectrum of quality of service guarantees. It is often the case, however, that flows must negotiate for services and quality attributes in real time. This is the basic FSQ model, where attribute requirements associated with flows are dynamically reconciled with network capabilities.

Flow Structures are an important element of the System Survivability Analysis (SSA) technique developed by the SEI CERT Coordination Center [Ellison et al. 1999a, Ellison et al. 1999b, Linger et al. 2000, Mead et al. 2000]. SSA is a structured engineering process aimed at improving survivability characteristics of new or existing systems. It has been applied to a number of government and commercial systems with good results. The SSA process identifies mission-essential system flows, that is, flows that must be available no matter what the threat environment and state of compromise. These flows of service uses are traced through the system architecture to reveal corresponding essential components. Next, representative intrusions are identified based on analysis of the threat environment, and expressed as flows for tracing through the architecture to reveal compromisible components. With this information it is possible to identify softspot components that are both essential and compromisible, followed by survivability analysis for improvements to resistance, recognition, and recovery strategies within the system architecture.

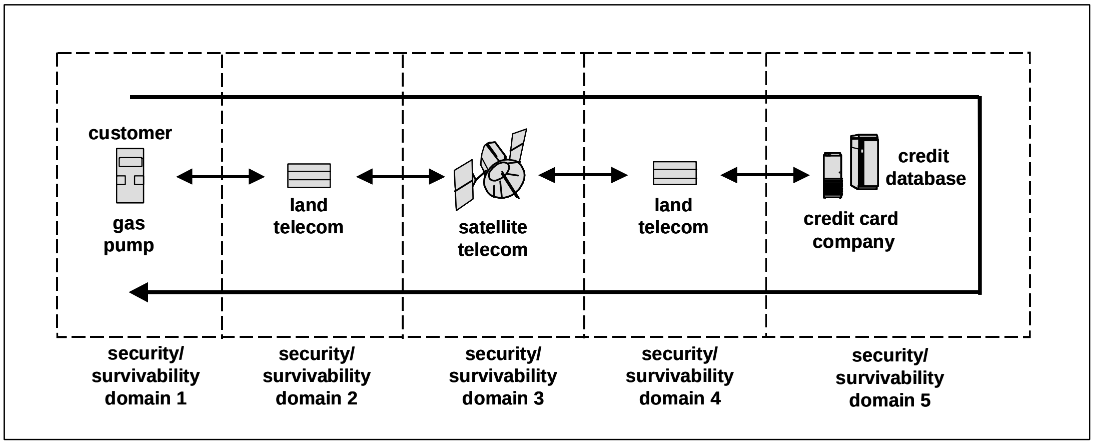
customer
gas pump
land telecom
satellite telecom
land telecom
credit database
credit card company
security/ survivability domain 1
security/ survivability domain 2
security/ survivability domain 3
security/ survivability domain 4
security/ survivability domain 5

Figure 14: Flow Security and Survivability Domain Traversals

These six engineering operations illustrate some basic uses of Flow Structure concepts and techniques. Additional engineering operations include derivation of logical network structures from FlowSets, and definition of statistical usage models of network systems for test case generation based on FlowSet usage patterns and probabilities.

5 Computational Quality Attributes

Unless a system is implemented on a closed internal network, many system details may be unknowable. Nevertheless, enterprises place extraordinary demands on systems for reliability, availability, security, and other key quality attributes [Haeckel 1999]. Substantial effort has been devoted to development of descriptive and often subjective characterizations of quality attributes, for example, survivability attributes [Ellison et al. 1999c, Sullivan et al. 1999]. Rather than focusing on descriptive methods, the Computational Quality Attributes (CQA) approach defines, computes, and acts upon quality attributes as dynamic characteristics of system operation. This section describes mathematical foundations and frameworks for these operations.

Many researchers have addressed component-based quality attributes, such as reliability, from the perspective of the system as a whole [Siegrist 1988, Krishnamurthy and Mathru 1997, Gokhale and Trivedi 1998, Yacoub et al. 1999, Hamlet et al. 2001]. However, from the perspective of a user of a distributed system, there is no need for a system view of quality attributes. The user is concerned with the provision of essential services, not the state of the system. The CQA approach addresses the user's concern. Quality attributes are defined at the service level, composed to the service and flow levels, and evaluated at either the service or flow level, or at both levels, depending on the user-specified CQA request. This CQA approach, illustrated in Figure 15, provides a consistent semantic framework for acquiring quality attribute performance information for essential flows and services, and for computing system quality capabilities for run-time management of flow execution.

5.1 The CQA approach

CQAs are defined as a functional mapping of usage (i.e. the input domain for a particular use, environment, and time) into attribute values that represent a measure of quality. This approach supports the description of any set of quality attributes and any models used to describe each attribute, provided each model yields a numeric value.

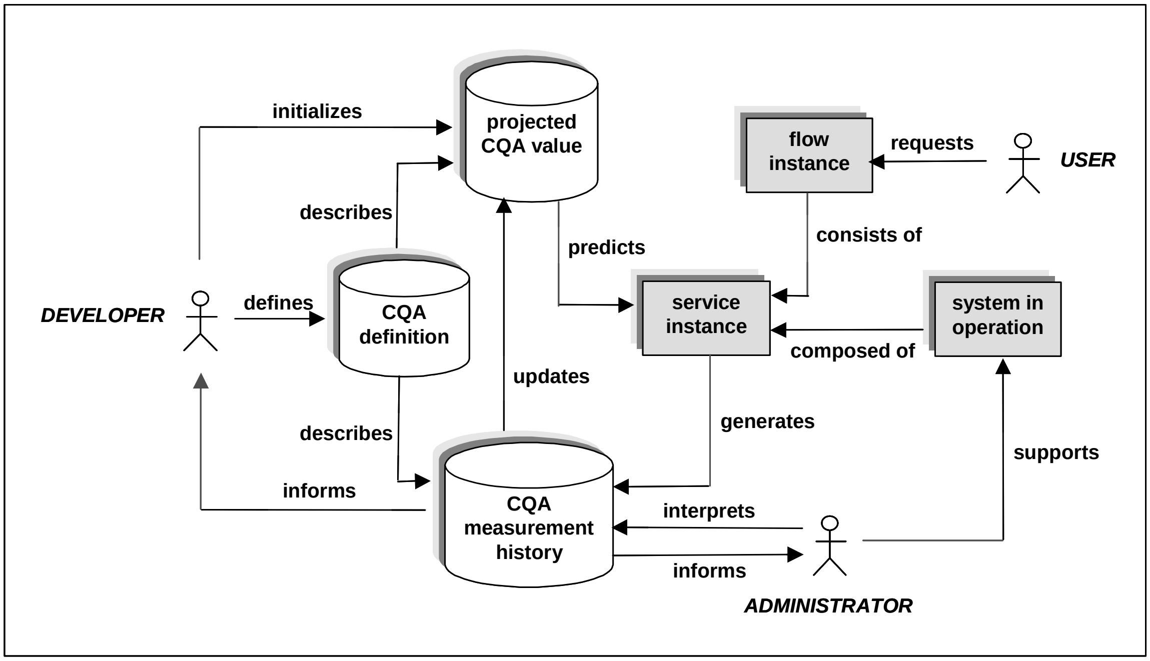

Figure 15: The Computational Quality Attribute Approach

The developer defines CQAs of interest and initializes projected CQA values for each service based on assertions from vendors, best estimates, historical records, etc. During operational use of the system, the following activities are iteratively performed as services are executed:

- Service executions are monitored and their quality attributes are measured.
- The quality attribute values are accumulated.
- The accumulated history is analyzed and used to update projected CQA values as appropriate. (Note that a single observation for a CQA provides almost no information. However, over time, the cumulative weight of the execution histories informs the methods for predicting CQAs for service instances.)

The user requests flows of services to be performed by the system. Flow requests include required levels of CQAs for the requested flow, as well as for execution of the individual services they reference. Thus, user-requested CQAs define constraints for flow execution.

The system administrator uses the CQA history to support decisions concerning updates to the distributed systems (e.g. load balancing and replicated services). The system administrator's knowledge of the state of the system and its services can provide valuable insights to support interpretation of the CQA history.

Because each service's CQAs can be calculated independently of other services and system components, both the implementation of the component and the distribution of the instances of the component can provide a variety of opportunities to achieve desired values for CQAs of interest. Multiple choices may be available to the designer of a component for implementation of a service, each choice with different values for quality attributes. In addition, the system administrator may distribute one or more instances of a particular implementation of a service across a network, where each distribution may yield different values for CQAs.

5.2 CQA Definition

The general model for definition of a CQA, q_i, is a hierarchy of functional definitions:

$q_i = f(q_{i,1}, q_{i,2}, ...)$, where $q_{i,1} = g(q_{i,1,1}, q_{i,1,2},...)$

Each CQA has a precise meaning and a formal functional representation. In general, the process of CQA definition can be described as follows:

- Determine the quality attributes of interest.
- Recursively define the functions for a quality attribute as a hierarchy of functional definitions of attributes until each leaf attribute has been defined as a computational function that maps usage into a non-negative real number that represents a measure of quality.
- Determine the data structure and procedures for storing the attribute values.

Note that measuring and storing CQA values at detailed levels can adversely impact system performance. Thus, tradeoff analyses are typically required to determine the appropriate granularity of the values to be stored. Issues include frequency of calculation, and whether to store actual or average values.

The approach to CQA definition accommodates all degrees of complexity in quality attribute definition. Any number of attributes may be of interest to users of a distributed system, including attributes that describe behavioral requirements and attributes that describe performance characteristics. Some attributes (such as availability, reliability, and response time) have a rich literature and can be easily defined as CQAs. Other attributes will require more effort to develop a CQA definition. For example: survivability may be defined as a function of system architecture and other attributes; system architecture can be defined as a function of a variety of attributes including connectivity; etc. Some example CQA definitions are provided in [Walton et al. 2002].

5.3 Flow Request Analysis

In order to implement the CQA framework to support distributed system use, a system's services must be "attribute-enabled" for the CQAs of concern to users. That is, a mechanism must exist to support CQA evaluation and reporting. This mechanism may be implemented directly at the level of the service, or computed by evaluation and composition of CQAs from lower-level services. A CQA for which a service is not attribute-enabled will yield a value of "0" if any flow request includes a constraint on that attribute.

Lower level CQAs are composed to establish the value of the CQAs on the next higher level. CQAs are also composed across services to establish flow properties. For the sake of simplicity, the CQA approach assumes service executions are independent. (To assume otherwise would require knowledge of the designs and implementations of each of the components that make up each service.) Because CQAs map usage into attribute values, it is important that each CQA value be determined based on the domain of interest to the usage flow.

Given a constrained flow request and a set of candidate flows, the goal of flow request analysis is to determine whether each candidate flow satisfies the set of CQA constraints. If each CQA constraint is defined as a minimum acceptable value, then the set of acceptable flows resides within a convex region of acceptable quality as defined by the CQA constraints. Details of CQA composition and flow request analysis are provided in [Walton et al. 2002]. A high level description and a simple example are provided here to illustrate the concepts.

A user specifies a flow request as follows:

1. Define the flow as a composition of system services that carry out user tasks.
2. Determine the CQAs of interest for this flow request based on the input domain for this particular use, environment, and time.
3. Specify constraints for the requested services and constraints for the flow in terms of acceptable values or ranges of values for CQAs.

A flow request is processed as follows:

1. Candidate flows are identified that provide the requested composition of system services.
2. Each candidate flow is evaluated based on whether the set of projected CQA values associated with the services included in the candidate flow satisfy the flow request's CQA constraints. (If multiple candidate flows satisfy the flow request, negotiation with the user may be performed to select the optimal flow based on user-specified priorities.)

Candidate flows are evaluated as follows:

1. In addition to requested services, represent all network connections as services.
2. Predict the values for each CQA of interest for each service in the flow. (If a service is not attribute-enabled for a CQA of interest, assign a value of 0 to that CQA.)
3. Check the CQA constraints specified for each service by comparing the predicted CQA values to the constraints. If all of the service-level constraints are satisfied, continue to the next step. Otherwise return a "no" response to the request.

4. Check the flow-level CQA constraints:

   a. Develop a state-based graphical model that includes all possibilities for satisfaction of the CQA constraints, including adverse results.

   b. Convert each CQA into a probability distribution and annotate the arcs of the graphical model with the probability distribution, yielding a probabilistic model. (Use weights as needed in the situations where the CQA units are not uniform across all the services.)

   c. Compute the projected flow-level CQA based on the probabilistic model.

5. If the projected flow-level CQA satisfies the flow-level constraints, return "yes". Otherwise return "no".

5.4 A CQA Example

Consider the simple flow request illustrated by Figure 16. The user has requested a flow instance that consists of the execution of an instance of service A followed by execution of an instance of service B. The CQA q_1 constrains the flow request. q_1 is defined as the predicted reliability, represented by a real number in the interval [0,1], and interpreted as the probability that a single execution will not fail.

The following CQA constraints are specified:

- Service-level constraints: $q_1 > 0.97$ for the instance of service A; and $q_1 > 0.96$ for the instance of service B
- Flow-level constraint: $q_1 > 0.94$ for the entire flow instance.

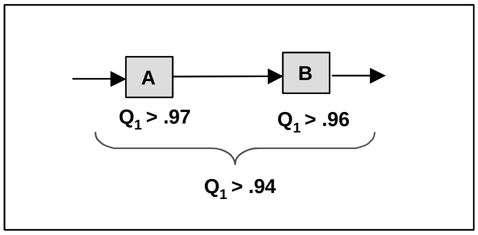

Figure 16: Simple Constrained Flow Request

Representing the connections as services Con1, Con2, and Con3, assume that a candidate flow is identified with the following CQA predictions:

q_1 = .99 for Con1; q_1 = .98 for A and Con2; q_2 = .97 for B and Con3.

These predicted CQAs satisfy all the service-level constraints specified by the user. Thus, the next step is to determine whether the candidate flow satisfies each flow-level constraint.

Figure 17 illustrates the state diagram used to analyze the flow-level CQA constraint q_1 = 0.95 for the candidate flow. Each service is represented by a state with two exit arcs: fail to satisfy the CQA constraint and continue to the next service. Continue occurs if the service satisfies the specified CQA constraint. Failure occurs if the service fails to satisfy the CQA constraint. The probabilities associated with the exit arcs for each service sum to 1.0. In this example, if one assumes independence in the probabilities that each service will not fail, the probability that the candidate flow will execute without failure is the product of the predicted q_1 values for each service. At the flow-level, predicted q_1 is calculated for this candidate flow as follows:

predicted q_1 = 0.99 * 0.98 * 0.98 * 0.97* 0.97 = 0.89

Thus, the candidate flow does not satisfy the flow-level constraint q_1>.94.

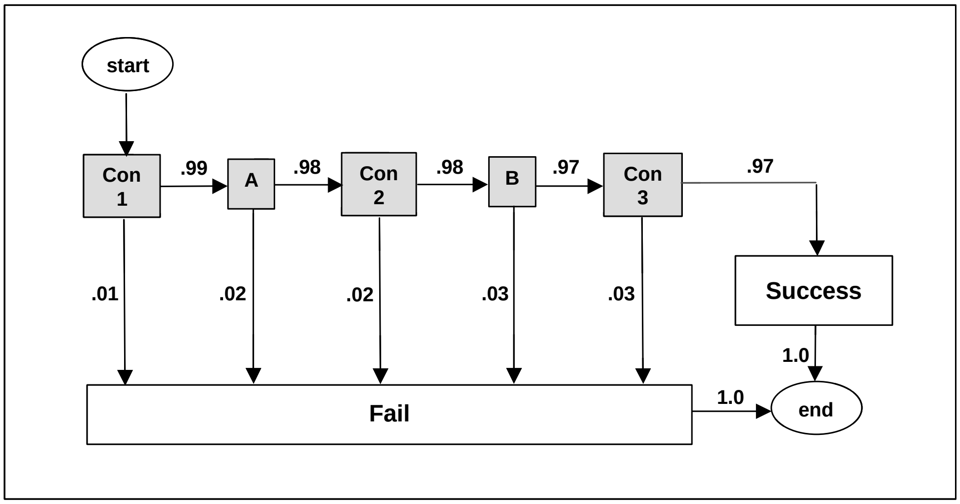

Figure 17: State Diagram for Analysis of q_1

5.5 Considerations for CQA Analysis

Transient faults and considerations of dynamic changes in the CQAs across the input domain can cause composition to be problematic, requiring the use of policies and assumptions such as the following:

- If a resource is not available when it is needed by a flow instance, the flow instance will abort without waiting for the resource.
- Flows cannot reserve resources.
- Flow execution does not include look-ahead.

- No repairs or replacements are done during a flow.
- Component and service failures are statistically independent.
- Probability distributions associated with the CQAs do not change during the execution of a flow.

An understanding of both use and replication is required before composition of CQAs can be performed. For example, if three nodes are each executing the same service on the same data, failure of a node is not a problem unless all three replicated nodes fail. In contrast, if three nodes execute the same service on different data, a failure in a single node causes a failure in the flow requiring execution of the service on that node's data.

5.6 Dynamic Updates of Computational Quality Attributes

The algorithm for updating projected CQAs from the history must be sensitive to the fact that the system is continually evolving. Replacement or maintenance of a node, communication link, or service may change the system properties drastically, but the user may have no way to know that the change occurred. Each use of the system may in fact be executing on a different version of the software or in a different usage environment (for example, a different network configuration) for which the history is not applicable. Information about system evolution may not be directly available. Traditional statistical inference techniques are inadequate to support measuring and prediction of CQA values in these situations.

To estimate a value for a CQA for an evolving system for which the system state is unknowable requires a method that makes the best use of every source of available relevant evidence, including information obtained from vendors, personal judgment, accumulated history, and knowledge of the occurrence of specific events (updates to a component, etc.) that may invalidate the history. At present such network-based applications are often adjusted manually based on intuition and incomplete knowledge of current system state. The CQA approach replaces these informal techniques by using Bayesian statistical methods to provide a systematic response-based approach to estimation of CQA values. Bayesian statistical methods provide a mathematical framework that allows representation of everything known (or assumed) about a CQA in a simple functional form, and support updating this functional form with new knowledge as it becomes available [Lee 1989, Royall 1997].

A Bayesian probability is a formalism that supports reasoning about beliefs under conditions of uncertainty and using disparate sources of evidence. Unlike classical statistical inference, the Bayesian approach considers the observations to be fixed, and the parameters to be random variables with their own statistical distributions. The Bayesian approach starts with a suitable set of pre-existing beliefs (the "prior distribution"). As new data become available, they are combined with the prior distribution to obtain a new (posterior) distribution that can be used to support analysis.

When there is no credible pre-existing evidence, the priors must be based on professional judgment. For example, for critical system use, one might use CQA prior values that represent the worst case until evidence is available that indicates that the situation is less grim. (Start with a prior belief that availability = 0 for a service, where availability is the probability that the service will be available when needed. If evidence accumulates that the service is indeed consistently available when needed, the prior belief will become increasingly irrelevant, and the estimated value for availability should be increased.)

The Bayesian approach to dynamic updates of CQAs provides several advantages. It

- Allows both predicted and field/test data to be combined into a precise and convenient function for the CQA.
- Makes available all knowledge of CQA values.
- Allows easy updates to the knowledge as appropriate.

However, the Bayesian approach assumes independent risk factors and therefore tends to overestimate risk when there are multiple correlated risk factors. In addition, the dynamic nature of the CQAs and possible correlation between the CQAs can make it difficult to validate the CQA measurements. To further complicate matters, to evaluate functional CQAs, flow executions must be treated as statistically independent trials, ignoring potential issues such as internal system state and data stores. Thus, care must be taken to ensure:

- Sufficiently conservative priors are assigned
- Sufficient experience is gained before it is incorporated into the posterior distribution
- Each QA function is periodically reevaluated and reinitialized based on history and any knowledge about changes to the distributed system.

This Bayesian approach for dynamically updating CQA projections appears to be far better than the alternatives, given that complete knowledge of current and future system state is not possible. Mathematical details and examples of this approach are provided in [Walton et al. 2002]. The methods described here, namely the functional attribute model that maps service usage into attribute values, the state transition model for evaluating attributes, and the Bayesian methods for dynamic attribute updates, constitute a framework for Computational Quality Attribute research. Work is required to develop attribute-specific models within this framework. The requirement that attributes be measurable in a defined metric for computational purposes also permits human understanding and analysis not otherwise possible.

6 Flow Management Architectures

As noted earlier, Flow Structures and Computational Quality Attributes support system architectures that carry out dynamic flow and attribute management in execution. Flow Management Architectures can provide design and implementation frameworks for this purpose, as well as associated engineering processes for system architecture development. We envision an evolving, open family of FMA frameworks for architecture development both in the small and in the large.

FMA templates in the small can define topologies and functional capabilities that satisfy quality attributes for localized flow management. For example, a quality attribute could require network isolation of a particular class of flows for security purposes. Such a requirement often exists for external user flows that access enterprise Web sites. The FMA template in this case could define a DMZ-based topology to isolate external users from enterprise networks, plus functional isolation of operational and developmental Web servers. Administrative flows would also appear in the relevant FlowSet, and the template would include topological and functional capabilities for monitoring and controlling Web site operation. Note in this illustration that quality attributes involving flow isolation are imposed by the executing enterprise and not by the flow originators. It is invariably the case that the network services traversed by flows implement their own attributes and management procedures for processing incoming flows.

FMA templates in the large can define system-level topologies and functional capabilities for managing user-requested flow instantiations and reconciling attribute requirements where possible with service capabilities in real time operation. Such templates can accommodate traffic projections, geographic factors, communication patterns, and a host of other factors that drive large-scale network design.

FMA frameworks can also include engineering processes for mapping FlowSet specifications into network and service designs. FlowSets define access requirements for logical connectivity among services as a sufficient network topology, as well as functional requirements for the services themselves. Quality attributes associated with FlowSets impose additional requirements and constraints on network design. These relationships between network usage embodied in flows and attributes and network design embodied in connectivity and functionality provide an opportunity to develop systematic engineering practices for network development and validation.

7 Conclusion

Identification of flow, service, and quality as first-class concepts for the development of large-scale, network-centric systems is important for achieving unification of this complex engineering activity. Theoretical foundations developed in this research can prescribe engineering practices that will improve system management, acquisition, analysis, development, operation, and evolution. The following observations summarize this research vision.

- FSQ engineering supports complexity reduction and survivability improvement in development and operation of large-scale network systems composed of any mix of newly developed and COTS components.
- FSQ engineering provides systematic, scale-free semantic structures for requirements, specification, design, verification, and implementation.
- FSQ engineering supports seamless decomposition from user task flow, service, and quality attribute requirements to flow, service, and quality attribute implementations, with intrinsic traceability.
- User flows of services and quality attributes permit system development in terms of user views of services, as opposed to strictly functional decomposition or object-based composition.
- Flow Structures are deterministic for human understanding and analysis, despite the asynchronism of network behavior, thus enabling compositional methods of refinement, abstraction, and verification.
- Flow Structures reflect the realities of network-centric systems in dealing the Uncertainty Factors, to support enterprise risk management and system survivability.
- Flow Structures support definition of attack and intrusion flows for assessing system vulnerabilities and compromises, as a basis for security and survivability improvements.
- Computational Quality Attributes reflect the realities of network-centric systems, in assessing and reconciling quality requirements and capabilities as an intrinsically dynamic process.
- Computational Quality Attributes provide a scale-free, computational usage-centric (rather than system-centric) view of quality.
- Flow Management Architectures provide systematic and uniform methods for managing user flow instantiation and quality attribute satisfaction in execution.
- Foundations of Flow Structures can stimulate research on representation and analysis of flows at the requirements level within enterprises, and at the implementation level within system architectures.

- Foundations of Computational Quality Attributes can stimulate research in modeling and dynamic evaluation of important quality attributes and metrics.

FSQ research and development efforts will continue to explore foundations for Flow Structures, Computational Quality Attributes, and Flow Management Architectures. Several papers are in preparation that detail the mathematical foundations of Flow Structures and Computational Quality Attributes [Pleszkoch et al. 2002, Walton et al. 2002]. This work will provide a basis for engineering practices and support tools.

8 References

[Ellison et al. 1999a] Ellison, R.; Fisher, D.; Linger, R.; Lipson, H.; Longstaff, T.; & Mead, N. "Survivable Network System Analysis: A Case Study." *IEEE Software 16*, 4 (July/August, 1999): 70-77.

[Ellison et al. 1999b] Ellison, R.; Fisher, D.; Linger, R.; Lipson, H.; Longstaff, T.; & Mead, N. "Survivability: Protecting Your Critical Systems." *IEEE Internet Computing 3*, 6 (November/December, 1999).

[Ellison et al. 1999c] Ellison, R.; Fisher, D.; Linger, R.; Lipson, H.; Longstaff, T.; & Mead, N. *Survivable Network Systems: An Emerging Discipline* (CMU/SEI-97-TR-013, ADA341963). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1997. <http://www.sei.cmu.edu/publications/documents/97.reports/97tr013/97tr013abstract.html>

[Gokhale and Trivedi 1998] Gokhale, S. & Trivedi, K. "Dependency Characterization in Path-based Approaches to Architecture-based Software Reliability Prediction." *Proceedings of the 1998 IEEE Workshop on Application Specific Software Engineering and Technology*. Richardson, Texas, March 26-28, 1998. Los Alamitos, CA: IEEE Computer Society Press, 1998.

[Haeckel 1999] Haeckel, S. *Adaptive Enterprise: Creating and Leading Sense-and-Respond Organizations*. Boston: Harvard Business School Press, 1999.

[Hamlet et al. 2001] Hamlet, D.; Mason, D.; & Woit, D. "Theory of Software Reliability Based on Components." *Proceedings of the 23rd International Conference on Software Engineering (ICSE 2002).* Toronto, Canada, May 2001. Los Alamitos, CA: IEEE Computer Society Press, 2001.

[Hausler et al. 1990] Hausler, P.; Pleszkoch, M.; Linger, R.; & Hevner, A. "Using Function Abstraction to Understand Program Behavior." *IEEE Software 7*, 1 (January 1990): 55-63.

[Hevner et al. 2001] Hevner, A.; Linger, R.; Sobel, A.; & Walton, G. "Specifying Large-Scale, Adaptive Systems with Flow-Service-Quality (FSQ) Objects." *Proceedings of the 10th OOPSLA Workshop on Behavioral Semantics*. Tampa, Florida, October 2001.

[Hevner et al. 2002] Hevner, A.; Linger, R.; Sobel, A.; & Walton, G.. "The Flow-Service-Quality Framework: Unified Engineering for Large-Scale, Adaptive Systems." *Proceedings of the 35th Annual Hawaii International Conference on System Science (HICSS35)*. Hawaii, 2001. Los Alamitos, CA: IEEE Computer Society Press, 2002.

[Hoffman and Weiss 2001] Hoffman, D. & Weiss, D. *Software Fundamentals: Collected Papers by David L. Parnas*. Upper Saddle River, NJ: Addison Wesley, 2001.

[Krishnamurthy and Mathru 1997] Krishnamurthy, S. & Mathru, A. "On the Estimation of Reliability of a Software System Using Reliabilities of its Components." *Proceedings of Eighth International Symposium on Software Reliability Engineering (ISSRE '97)*. Albuquerque, New Mexico, November 1997. Los Alamitos, CA: IEEE Computer Society Press, 2002.

[Lee 1989] Lee, P. *Bayesian Statistics: an Introduction*. New York: Oxford University Press, 1989.

[Leymann and Roller 2000] Leymann, F. & Roller, D. *Production Workflow: Concepts and Techniques*. Upper Saddle River, NJ: Prentice-Hall PTR, 2000.

[Linger et al. 1979] Linger, R.; Mills, H.; & Witt, B. *Structured Programming: Theory and Practice*. Reading, MA: Addison-Wesley, 1979.

[Linger et al. 2000] Linger, R.; Ellison, R.; Longstaff, T.; & Mead, N. "The Survivability Imperative: Protecting Critical Systems," *Crosstalk 13*, 10 (October 2000): 12-15.

[Mead et al. 2000] Mead, N; Ellison, R.; Linger, R.; Longstaff, T.; & McHugh, J. *Survivable Network Analysis Method* (CMU/SEI-2000-TR-013 ADA383771). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2000. <http://www.sei.cmu.edu/publications/documents/00.reports/00tr013.html>

[Mills et al. 1986] Mills, H.; Linger, R; & Hevner, A. *Principles of Information System Analysis and Design*. San Diego: Academic Press, 1986.

[Mills 1988] Mills, H. "Stepwise Refinement and Verification in Box-Structured Systems." *IEEE Computer 21*, 6 (June 1988): 23-36.

[Mills and Linger 2002] Mills, H. & Linger, R. "Cleanroom Software Engineering." *Wiley Encyclopedia of Software Engineering: Second Edition*. New York: Wiley, 2002.

[Moore et al. 2001] Moore, A.; Ellison, R.; & Linger, R. *Attack Modeling for Information Security and Survivability* (CMU/SEI-2001-TN-001, ADA388771). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2001. <http://www.sei.cmu.edu/publications/documents/01.reports/01tn001.html>

[Parnas and Wang 1994] Parnas, D. & Wang, Y. "Simulating the Behavior of Software Modules by Trace Rewriting Systems." *IEEE Transactions on Software Engineering 19*, 10 (October 1994): 750-759.

[Pleszkoch et al. 1990] Pleszkoch, M.; Hausler, P.; Hevner, A.; & Linger, R. "Function-Theoretic Principles of Program Understanding." *Proceedings of the 23rd Annual Hawaii International Conference on System Science (HICSS23)*. Los Alamitos, CA: IEEE Computer Society Press, Los Alamitos, CA, 1990.

[Pleszkoch et al. 2002] Pleszkoch, M.; Linger, R.; Walton, G.; & Hevner, A. "Semantic Foundations for Flow Structures." (to be published)

[Prowell et al. 1999] Prowell, S.; Trammell, C.; Linger, R.; & Poore, J. *Cleanroom Software Engineering: Technology and Practice*. Reading, MA: Addison Wesley, 1999.

[Royall 1997] Royall, R. *Statistical Evidence: a Likelihood Paradigm*. New York: Chapman & Hall, 1997.

[Schneider 1999] Schneider, F. (ed.). *Trust in Cyberspace*. Washington, DC: National Academy Press, 1999.

[Shaw 1996] Shaw, M. "Truth vs. Knowledge: The Difference Between What a Component Does and What We Know It Does." *Proceedings of the 8th International Workshop on Software Specification and Design*. Berlin, Germany, March 1996. Los Alamitos, CA: IEEE Computer Society Press, 1996.

[Siegrist 1988] Siegrist, K. "Reliability of Systems with Markov Transfer of Control." *IEEE Transactions on Software Engineering 14*, 9 (September 1988): 1049-1053.

[Sikora and Shaw 1998] Sikora, R. & Shaw, M. "A Multi-Agent Framework for the Coordination and Integration of Information Systems." *Management Science 44*, 11 (1998): S65-S78.

[Sullivan et al. 1999] Sullivan, K.; Knight, K.; Du, X.; & Geist, S. "Information Survivability Control Systems," *Proceedings of 21st International Conference on Software Engineering*. Los Alamitos, CA: IEEE Computer Society Press, 1999.

[Walton et al. 2002] Walton, G.; Hevner, A.; Linger, R.; & Pleszkoch, M. "Computational Quality Attributes for Distributed System Operation." (to be published)

[Yacoub et al. 1999] Yacoub, S.; Cukic, B.; & Ammar, J. "Scenario-Based Reliability Analysis of Component-Based Software." *Proceedings of the 18th IEEE Symposium on Reliable Distributed Systems*. Switzerland, October 1999. Los Alamitos, CA: IEEE Computer Society Press, 1999.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY	2. REPORT DATE June 2002	3. REPORT TYPE AND DATES COVERED Final

4. TITLE AND SUBTITLE

Flow-Service-Quality Engineering: Foundations for Network System Analysis and Development

5. FUNDING NUMBERS

F19628-00-C-0003

6. AUTHOR(S)

Richard C. Linger, Mark G. Pleszkoch, Gwendolyn Walton, Alan R. Hevner

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

8. PERFORMING ORGANIZATION REPORT NUMBER

CMU/SEI-2002-TN-019

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

HQ ESC/XPK
5 Eglin Street
Hanscom AFB, MA 01731-2116

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12A DISTRIBUTION/AVAILABILITY STATEMENT

Unclassified/Unlimited, DTIC, NTIS

12B DISTRIBUTION CODE

13. ABSTRACT (MAXIMUM 200 WORDS)

Modern society could hardly function without the large-scale, network-centric information systems that pervade government, defense, and industry. As a result, serious failures or compromises carry far-reaching consequences. These systems are characterized by changing and often unknown boundaries and components, constantly varying function and usage, and complexities of pervasive asynchronous operations. Their complexity challenges human intellectual control, and their survivability has become an urgent priority. Engineering methods based on solid foundations and the realities of network systems are required to manage complexity and ensure survivability. Flow-Service-Quality (FSQ) engineering is an emerging technology for management, acquisition, analysis, development, evolution, and operation of large-scale, network-centric systems. FSQ engineering is based on Flow Structures, Computational Quality Attributes, and Flow Management Architectures. These technologies can help provide stable engineering foundations for the dynamic and often unpredictable world of large-scale, network-centric systems. Flow Structures define enterprise mission task flows and their refinements into uses of system services in network traversals. Flows are deterministic for human understanding, despite the underlying asynchronism of network operations. They can be refined, abstracted, and verified with precision, and deal explicitly with Uncertainty Factors, including uncertain commercial off-the-shelf functionality and system failures and compromises. Computational Quality Attributes go beyond static, a priori estimates to treat quality attributes such as reliability and survivability as dynamic functions to be computed in system operation. Computational Quality Attribute requirements are associated with flows and can be dynamically reconciled with network service attributes in execution. Flow Management Architectures include design and implementation frameworks for dynamically managing flows and attribute requirements, as well as processes for their development. FSQ foundations are defined by theorems that illuminate engineering practices and automation opportunities.

14. SUBJECT TERMS

Flow-Service-Quality, Flow Management Architectures

15. NUMBER OF PAGES

53

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL